COMP3211/9211 2003 Assignment 2

MM Cache

Contribution to final assessment: 10%

Due: 4.30 PM, Wednesday, 8 October

Overview

The goal of this assignment is to recommend a cache design for a specialized computer system designed primarily for performing high-speed matrix multiplications.

In order to fulfil this goal you are required to simulate the memory references of a straightforward (simple) matrix multiplication algorithm when implemented on a variety of machines with varying cache organizations. The simulation requires you to count the number of cache hits and misses and to assign costs to each type depending upon the design of the cache, the memory access pattern, and processor speed. The results of this investigation are then to be analysed and discussed before making a recommendation as to the most suitable organization for the assumed environment.

Detailed Requirements

At the heart of straightforward matrix multiplication algorithms is a triple-nested loop as follows:

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            C[i,j] += A[i,k] * B[k,j]
        end do
    end do
end do
```

While it is possible to restructure this loop to improve performance, let us assume the memory references of the system we are to help design can be modelled by implementing the above loop as is, with the straightforward optimisation that accumulation of the partial sum `C[i,j]` is done in a register prior to storing the result. *Memory references* that result in cache searches, and possible cache misses are defined to occur whenever an array element `A[i,k]` or `B[k,j]` is read in the above loop. We shall assume all other temporary, including loop, variables can be stored in registers. We shall also assume that array elements are stored in 32-bit words, and that each matrix has 128 rows and columns, i.e., `n = 128` above. See below for further assumptions you should make to carry out your work.

You are required to explore the performance of a variety of cache organizations in terms of the *cost* of memory references as defined above. This cost is to be gauged by counting the *number of hits* and *misses* that occur for each of a variety of cache

organizations. As a minimum you need to investigate the performance with *direct-mapped, fully associative,* and *n-way set associative* organizations. Performance will vary as cache size, block size, and associativity are varied. You should therefore record the hit/miss counts with each of the following parameter settings:

1. Cache size: 16, 32, 64, 128, 256, 512, & 1024 words
2. Number of words per cache line (block): 1, 2, 4, 8, & 16 words
3. Set sizes (for n-way associativity): 2, 4, 8, & 16 blocks

Not all parameter combinations make sense. You are only required to vary the parameters and make a final recommendation within the specified cache sizes. This may mean not exploring all set sizes nor all possible block size for small cache sizes.

You are encouraged to write your own well-structured and commented simulator in your preferred programming language. This means encoding the matrix multiplication algorithm and implementing parameterised versions of the three cache organizations. Your simulator also needs to count the number of hits and misses that occur during the matrix multiplication loop. Note that you are only required to assess the number of hits and misses, not the implied delay to the running of the algorithm.

An alternative may be to find a cache simulator that allows you to investigate the problem as specified without much programming. Taking this route is only advisable if you have a great deal of experience modifying and building downloaded software, otherwise it may actually take more time and not succeed.

The hit to miss ratio is insufficient in this case to assess the relative merits of the above cache designs. These counts need to be scaled according to a number of further assumptions. Let us first assume that the target machine will have a clock frequency of 1 GHz. Furthermore, let us assume that the memory bus is constrained to be 4 words wide. Larger reads will therefore require multiple memory accesses, but these can be interleaved. The first access will require 50 cycles, and each successive read incurs a further 10 cycle penalty. Cache line sizes of 1 – 4 words therefore incur a miss penalty of 50 cycles, a size of 8 words leads to a miss penalty of 60 cycles, and 16 word organizations will incur a miss penalty of 80 cycles.

The hit time will vary as well for direct and n-way set associative caches as the size of the cache increases. For caches with 1 – 64 lines or sets, let us assume a hit time of 1 cycle. For sizes in the range 128 to 512, the hit time will be 2 cycles, and for 1024 lines, a hit time of 3 cycles should be used. You can assume that a fully associative cache always has a hit time of 1 cycle.

Further assumptions you should make

- Matrices are stored in memory in row-major order
- The machine is a load/store architecture, meaning matrix elements must be loaded into a register before they can be used
- There are sufficient registers for all temporary variables as well as the matrix elements of a single loop (for concreteness, let's say it is 16 registers)
- The compiler will not optimise the above algorithm save to make a single store to `C[i,j]` when it is has been calculated. This write should not be counted as a memory access
- The above miss penalties include the time to load a register from cache after a new line has been retrieved from memory.

- For the associative caches, assume a round-robin replacement policy – an index to the entry that is to be replaced is incremented modulo the number of cache lines/set size after each replacement. This index can be initialised to zero. Only one index is to be used for the set associative organizations.

Deliverables

You are required to produce a printed report that analyses and discusses your results prior to making a recommendation.

1) As a minimum, you are requested to tabulate the results for each cache organization. For example, for the direct-mapped cache, your table should include the following data in columns:

   a) Cache Size,
   b) Number of Words per Cache Line,
   c) Number of Hits,
   d) Hit Cost = Number of Hits x Cost per Hit (as defined in previous section),
   e) Number of Misses,
   f) Miss Penalty = Number of Misses x Cost per Miss (as defined above),
   g) Total Access Cost = Hit Cost + Miss Penalty

   The data should be tabulated in increasing cache size and number of words per cache line order.

   For the other organizations, the column headings may need to be adjusted, or additional information may need to be included to make a sensible presentation.

   To augment such tabulated data, and to aid the analysis and discussion of your results, you could consider plotting the data in a meaningful way.
   [30 marks]

2) Your analysis and discussion (separate sections) should cover the significant features of the results. Around half a page each should suffice once the data has been tabulated. [40 marks]

3) In an Appendix to your report you should include a listing of your program or details of how you configured any externally sourced simulator. Describe your experimental method in sufficient detail to be able to reproduce your results from your simulation methodology. [30 marks]

A printed copy of the above deliverables including a title page identifying

(i) the assignment title: COMP3211/9211 2003 Assignment 2,
(ii) your Tutorial day & time,
(iii) your Tutor's name, and
(iv) the names and student numbers of the members of your group (for this assignment, you are expected to pair with another person from your tutorial group)

should be submitted to and marked RECEIVED by the student office by 4.30 PM, Wednesday, 8 October, 2003.

You are also required to turnin an ASCII file containing the text of your report, including tables and appendices (program files, experimental method) using the command **`give a2-cache assign2.txt`** by 5.30 PM, Wednesday, 8 October, 2003.

Please refer to the course outline for an explanation of the penalties that apply for late or copied work.